# 第二部分 实战篇（第10～12章）

# 第10章 Angel Panda游戏的设计与实现

在掌握了第一部分基础篇的内容之后，读者已经可以制作一款包含各种元素的游戏。从本章开始，将进入实战部分的学习。

本书内容主要是关于Cocos2D开发的，既然要开发一款完整的游戏，仅仅懂得开发技术是远远不够，必须对游戏的策划和设计有一些基本的了解。

本章将对iOS游戏的策划与设计进行简要介绍，并简单介绍Angel Panda这款游戏的设计思路，以帮助开发者更好地了解如何在实际的项目开发中策划与设计一款游戏。

## iOS游戏完整的制作流程

当我们在iPhone或iPad上玩各种有趣的游戏时，当我们在大街小巷中忙碌穿行时，当我们在和朋友一起品尝美食和美酒时，甚至当我们在一天劳累过后的美梦中微笑时，经常会有灵光一闪的时刻，一个令自己都感到无比激动的想法出现了！如果可以把这样的想法做成类似Jetpack Joyride这样家喻户晓的游戏，该是多么令人兴奋的事情呢！

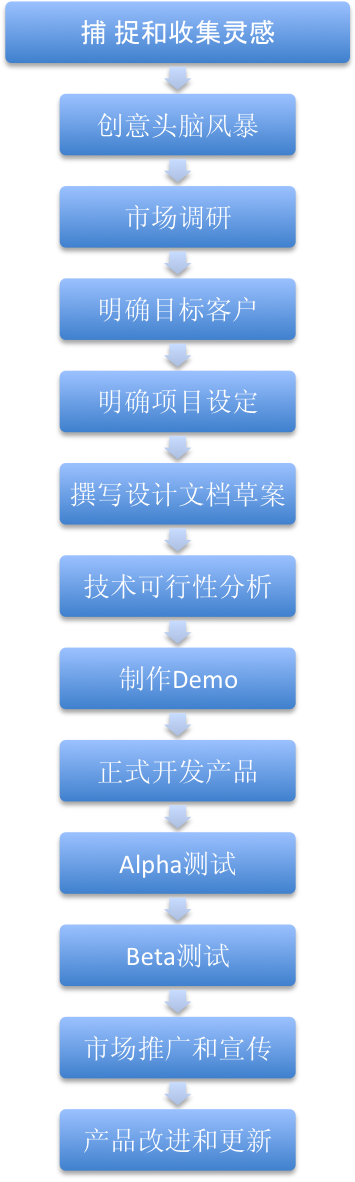
可是大多数时候，大多数人的想法只是停留在脑海中，直到这个弥足珍贵的想法如浮云一样淡淡散去。直到有一天，当自己习惯性地打开手机中的App Store时，突然发现在游戏排行榜上有一个颇受欢迎的游戏。当看完这款游戏的简介后，或许会像黄西面对美国历史课的考试一样发出一声感叹，“Ah!...”。当下载完游戏在设备中畅玩之后，又会发出另一声感叹，“Ah!...”。这省略号的内容就是，这个游戏不是我曾经的想法吗！可是，这是别人创作的游戏，而你的想法只是脑海中一块微不足道的记忆碎片而已！于是，一种淡淡的失落涌上心头。

事实上，在苹果推出App Store之前，很多人创造属于自己的游戏的梦想都是被这类叹息活生生地毁灭。的确，在App Store推出之前，无论是PC游戏还是主机游戏的开发，都需要昂贵的商业授权费用和推广费用，需要复杂的开发技术，以及庞大的团队支撑。而在App Store推出之后，由于移动游戏开发的简单易行，很多个人开发者和小型团队如同雨后春笋一样焕发生机，并取得了一定的成功。在这些团队之中，推出Angry Birds的芬兰团队Rivio可谓众人中的翘楚。自此以后，开发一款属于自己的游戏不再是天方夜谭，更不是痴人说梦，而成了触手可及的事实。

但是，如何把下一个“愤怒的小鸟”般伟大的想法变成一款实际的游戏呢？有很多事情要做，以下就是一款iOS移动游戏的完整制作流程。

1. 时刻提醒自己在生活中的点滴间记录与捕捉灵感，抓住所谓灵光一闪的瞬间。
2. 从自己平时收集到的灵感中筛选出最有可能制作成游戏的想法，并和自己的团队成员进行创意头脑风暴，挑选出一个想法作为游戏的基础。果是个人开发者，则需要在自己的多个想法间进行PK，但最终也必须明确选择其中的一个创意。此外，一个完整的移动游戏开发团队至少包含一名主程序、一名主美工和一名主策划。当然，如果是个人开发者，就必须身兼数职。
3. 对游戏创意进行市场调研。在App Store中寻找有没有类似的游戏，同时了解App Store中的用户偏好。充分考虑市场层面的因素，有些时候甚至需要果断放弃当前的创意，重新回到第二步。
4. 将想法高度概念化，并锁定自己的目标客户和目标国家市场。针对目标客户和目标国家市场的细分市场越明确越好。举例而言，如果目标客户锁定为18~35岁的铁杆Geek、白领、中高收入人群和喜欢科幻类电影的玩家，那么与科幻电影或小说密切相关的游戏创意，此类人群会备注关注。
5. 根据目标客户和目标国家市场确定项目，明确游戏的背景设定和主要美术风格。如果团队中有专门的音效师，则此时可以明确游戏的主体音乐风格。
6. 策划和设计人员根据游戏创意和目标客户撰写游戏的设计文档草案，并在必要时制作桌面游戏原型。
7. 邀请技术开发人员参与讨论，对游戏制作的技术可行性进行分析，并选定所需的开发技术和开发工具。
8. 根据设计文档草案及所选定的开发技术和工具制作简单的原型和演示版本。
9. 进行正式的产品开发阶段。此时游戏的关卡设计、美术资源和音效资源都需要同步进行。
10. Alpha测试阶段。此时可以对游戏的内容进行完善，同时替换美术和音效资源，并修正比较重大的bug。Alpha测试的主要参与人员是参与项目开发的人员。
11. Beta测试阶段。此时主要是对一些小的bug进行修复。Beta测试的主要参与人员可以是团队的全体成员，甚至可以邀请用户和媒体进行测试。在此阶段，应该充分做好市场推广的各项准备，可以请美术设计人员制作相关的宣传图片和视频。
12. 产品开发完成，准备上传到App Store。此时可以启动市场推广和宣传工作，并为下一个项目进行头脑风暴和想法PK。
13. 根据用户反馈或创意升级对游戏进行完善更新。

图10-1展示了一款iOS游戏的完整制作流程。



1. 一款iOS游戏的完整制作流程

了解了一款iOS游戏的完整制作流程后，开发者就可以着手自己或者和团队写作进入游戏的制作了。

## Angel Panda 游戏设计

游戏设计对于制作一款游戏来说是非常重要的环节。在进入实际的代码开发之前，项目开发团队的主要成员，特别是策划人员、程序人员和美术人员必须对游戏的策划方案达成一致。

本节将对Angel Panda的游戏设计进行简要的描述。

### Angel Panda 的创意来源

Angel Panda游戏的创意来源是HalfBrick出品的iOS经典休闲动作游戏Jetpack Ride，如图10-2所示。



1. Jetpack Joyride游戏界面

而Angel Panda（ 熊猫天使）游戏就是模仿Jetpack Joyride风格而设计的，目标用户为儿童和休闲玩家，同时由于熊猫形象的深入人心，目标国家没有特定的限制。

### Angel Panda 的故事背景及游戏规则

聪明勇敢的熊猫大侠Sam和家人一直在山谷村过着幸福而平静的生活，与世无争。直到有一天，人类占据了山谷村，把那里的森林砍伐殆尽，同时把原本居住在那里的可爱动物赶出来家园。然而没有任何一个种族可以始终统治地球，由于某种生化病毒的影响，人类不得不撤离了地球，而不受病毒影响的熊猫族得以再次回到故土家乡。

在Angel Panda这款游戏中，可爱的熊猫天使Baby受Sam所托，前往当年的山谷村探路。不过这一路上并非一帆风顺。虽然熊猫天使Baby可以在天空中飞翔，但人类残留的机器人对非人类生物仍然不会手下留情。而在地面上，则有着各种基因变异的植物怪兽。

简而言之，Angel Panda就是玩家扮演可爱的熊猫天使Baby返回故乡的休闲小游戏。

由于目标客户定位在儿童和休闲玩家上，Angel Panda游戏的美术风格将以卡通可爱和趣味性为主，而音效风格也是轻松而诙谐的类型。

Angel Panda的人物形象如图10-3所示。



1. 熊猫天使Baby

Angel Panda的游戏规则非常简单，玩家只需触碰屏幕，就可以让熊猫天使Baby飞翔；如果停止触碰屏幕，Baby就会在地面上行走。玩家需要注意躲避天空中的飞机、导弹等机器杀手，以及地面上的各种植物怪兽。

在这款简单的游戏中，玩家只需要安全飞行1分钟就可以抵达美丽的故乡山谷村。

作为一个示例项目，Angel Panda只设计了单一关卡。

### Angel Panda的游戏体验流程

在游戏设计阶段，需要明确玩家的完整体验流程。虽然Angel Panda游戏只是一款简单的单机小游戏，也要从用户体验的角度清晰列出游戏流程中的每个环节；

1. 玩家通过AppStore推荐、朋友推荐或宣传文章了解到这款游戏；
2. 玩家在查看游戏的图标、描述和截图后决定是否购买或下载游戏；
3. 玩家通过iTunes或设备，使用Wi-Fi或3G网络连接下载游戏；
4. 玩家打开游戏，看到加载画面；
5. 进入游戏主场景 ；
6. 触碰“开始”按钮开始游戏；
7. 玩家顺利完成游戏，或者失败。

## 开发前的准备工作

在正式进入游戏开发之前，要提前做好相关的准备。这些准备工作包括下载和安装合适的开发工具、安排游戏的开发流程，以及准备所需要的美术和音效素材等。

本节将对这些准备工作进行简单的介绍。

1. 下载并安装合适的开发工具

首先一台安装苹果Mac OS Snow Leopard、Mac OS Lion或Mac OS Mountain Lion操作系统的电脑。一般来说，只要是2009年之后购买的苹果笔记本系列或iMac都装有以上两个操作之一。如果是Snow Leopard操作系统，请升级到最新的10.6.8及以上版本；如果是Lion操作系统，请升级到最新的10.7.2及以上版本；如果是Mountain Lion操作系统，请升级到最新的10.8.2及以上版本。

在电脑和操作系统就位之后，安装Xcode开发工具包。由于Cocos2D v2.0以后只支持Xcode 4.x，因此请在电脑上下载安装最新的Xcode 4.4及以上版本。

接下来安装Cocos2D，本章代码采用Cocos2D v2.0开发，关于如何下载和安装Cocos2D请参考本书1.2节。

Angel Panda支持iOS 5.1和最新的iOS 6.0，编程开发工具是Xcode 4.4或Xcode 4.5，测试工具是Xcode中内置的Instruments，游戏开发引擎是Cocos2D v2.0，并支持最新的v2.1 beta2版本，游戏物理引擎是Box2D v2.2.1，编程语言以Objective-C语言为主，也包含少量的C++语言代码。

注意 Cocos2D v2.0之后的版本和之前的1.x版本存在不小的差别。如果开发者使用的是1.x版本，那么建议在学习本章时一定要下载安装v2.0之后的版本。否则可能会出现项目无法正常编译运行，或者出现未知bug的情况。

此外，本项目中所使用的其他开发工具还包括：

* Garage Band（编辑音效）
* Adobe Photoshop（美术资源）
* Zwoptex（精灵表单）
* Hiero（编辑字体）

以上就是开发Cocos2D游戏所需要的主要开发工具。

1. 将项目分解成不同的模块

在开发任何一个项目之前，必须对项目进行分解，这样既可以让开发过程有条不紊，也可以很好的和其他人进行协同开发。

Angel Panda游戏项目任务分解为以下六部分，在实际的开发过程中，每一部分可以由不同的开发人员来分别实现。

##### 第一部分：创建新项目

该部分的内容比较简单，主要是使用模板创建新项目，添加项目中所需的资源，并做好其他准备工作。

##### **第二部分：**设置游戏主场景

该部分主要放置背景精灵元素和标签等。

##### **第三部分：**添加游戏角色

该部分主要工作是把游戏角色添加到场景中。游戏角色包含玩家角色和敌人角色两部分。

##### **第四部分：**添加用户交互机制

该部分主要添加用户交互机制，比如本项目中的触摸事件处理。

##### **第五部分：**添加游戏逻辑

该部分在游戏中非常重要，其中实现了游戏的主要逻辑、游戏赢输机制，以及简单的碰撞检测等。

##### **第六部分：**添加音效

该部分内容是添加游戏中所需的音效。

本章将对项目分解后的六个部分进行讲解。

1. 准备游戏中所需要的资源

除了项目代码，游戏中还需要使用大量的美术、字体、音效和粒子效果等。

如果是团队开发，可以由美术人员和音效人员等分别准备所需的资源。如果是个人开发者，则需要使用本书中所提到的相关工具来自己进行准备。本章所需要的相关资源在cha10/Angel Panda/resources。

本章资源文件夹中所提供的美术资源包含针对Retina屏幕（iPod Touch 4、iPhone 4和iPhone 4S）的适合960\*640分辨率的图片，以及针对普通屏幕（iPhone 3GS、iPod Touch 3等设备）的适合480\*320分辨率的图片。比如项目中用到bg.png作为游戏的主场景图片，该图片针对普通屏幕的分辨率是480\*320。为了适应Retina屏幕的需要，应提供满足960\*640屏幕分辨率的相同图片，同时把文件加上后缀-hd，即最终的文件名为bg-hd.png。只要提供两种适合两种不同分辨率的图片，并为Retina屏幕准备的图片加上-hd后缀即可。通过这种方式，开发者无需修改任何代码就可以直接支持Retina屏幕的显示。

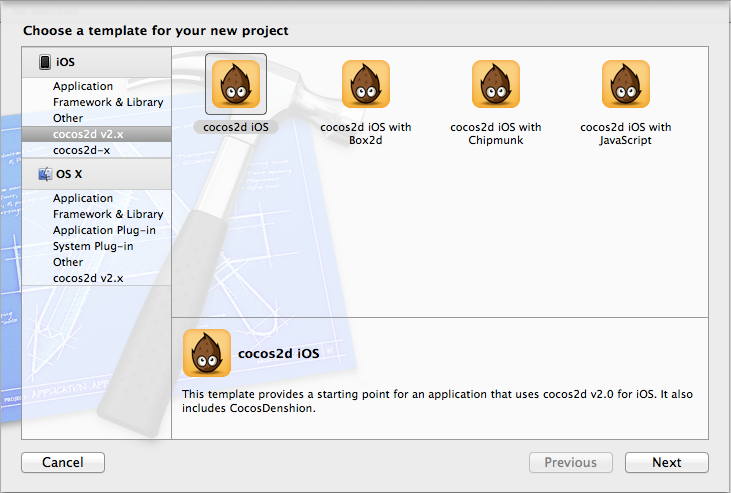
而如果是为iPad设备开发的游戏，则需要准备对应屏幕分辨率为1024\*768的图片。此外，对于最新的屏幕为4寸的iPhone 5或iPod Touch 6，则需要单独提供适合1136\*640分辨率的图片，以及名称为Default-568h@2x.png的加载图片。在程序代码中还需要利用UIDevice的信息以及屏幕尺寸大小来判断当前设备的分辨率，并提供对应的图片。

## Angel Panda游戏的实现

按照10.3节的任务分解，本节将分别实现主场景的设置、添加游戏角色、添加用户交互、添加游戏逻辑以及游戏中的音效。

1. 使用Cocos2D模板创建新项目

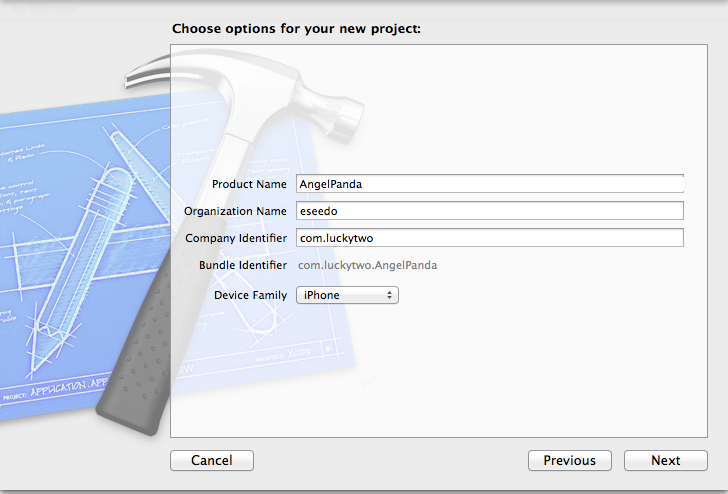
打开Xcode，单击菜单栏上的File→New→New Project以创建新项目，会看到4个Cocos2D模板，如图10-4所示。



1. 选择cocos2d iOS标准模板

提示 Cocos2D v2.1 beta1之前的版本只有3个模板， cocos2d iOS with JavaScript是v2.1版本所新增的模板。

选择第一个模板，给项目命名“AngelPanda”，其他保持默认选项，然后单击“Next”，选择要保存的目录，然后单击“Create”以创建项目。如图10-5所示。



1. 创建AngelPanda项目

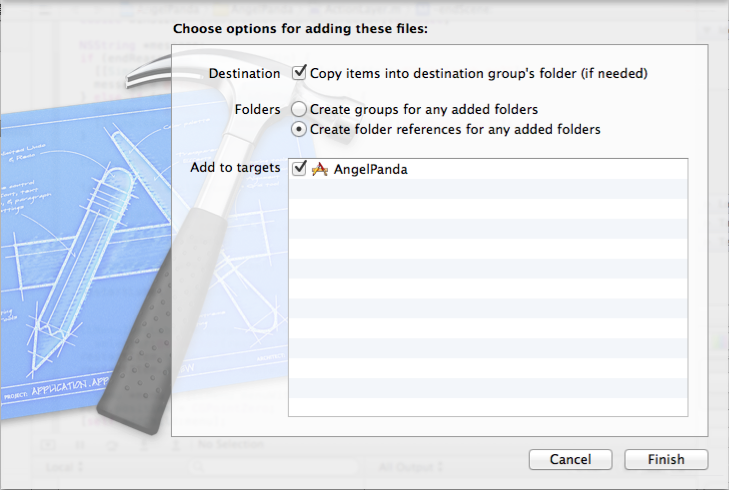
使用快捷键Command +R编译运行AngelPanda项目，就可以看到经典的Hello World画面，如图10-6所示。



1. 编译运行后在模拟器中显示的Hello World画面

#### 添加资源

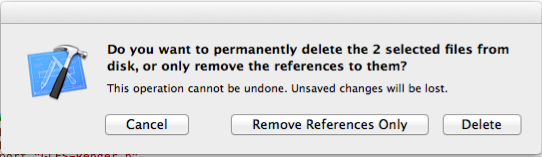
在创建完新项目之后，首先要把所需资源添加到项目之中。请从cha10/Angel Panda/resources中获取本章所需要的资源文件，直接拖到Xcode的AngelPanda项目中的Resources里，记住一定要选择“Copy items into destination group’folder（if needed）”，如图10-7所示。



1. 将资源文件添加到项目中

#### 从项目中删除模板中自带的文件

在Xcode中选中HelloWorldLayer.h和HelloWorldLayer.mm这两个文件，然后按下键盘上的Delete键，当出现图10-8所示的警告时，再次单击Delete来彻底删除这两个文件。



1. 从项目中删除HelloWorldLayer.h和HelloWorldLayer.mm

在Xcode中右键单击Angry Panda组，选择New File，然后选择iOS→Cocoa Touch→Objective-C class，选择Next。选择Subclass of NSObject，然后单击Next，将文件命名为ActionLayer.m，单击Save。

此时，ActionLayer.h和ActionLayer.m中几乎没有内容。

更改ActionLayer.h的代码如代码清单10-1所示。

1. 更改ActionLayer.h的代码

#import <Foundation/Foundation.h>

#import "cocos2d.h"

@interface ActionLayer: CCLayer{

}

+(id)scene;

@end

以上代码很简单，说明ActionLayer继承自CCLayer，并声明了一个类方法+(id)scene。

更改ActionLayer.m的代码如代码清单10-2所示。

1. 更改ActionLayer.m的代码

#import "ActionLayer.h"

@implementation ActionLayer

+(id)scene{

CCScene \*scene =[CCScene node];

ActionLayer\*layer = [ActionLayer node];

[scene addChild:layer];

return scene;

}

-(id)init{

if((self =[super init])){

CCLabelTTF \*label = [CCLabelTTF labelWithString:@"ANGEL PANDA" fontName:@"Noteworthy-Bold" fontSize:40];

CGSize winSize = [CCDirector sharedDirector].winSize;

label.position = ccp(winSize.width/2,winSize.height\*0.85);

[self addChild:label];

}

return self;

}

@end

在以上代码中，分别实现了scene类方法和init初始化方法。

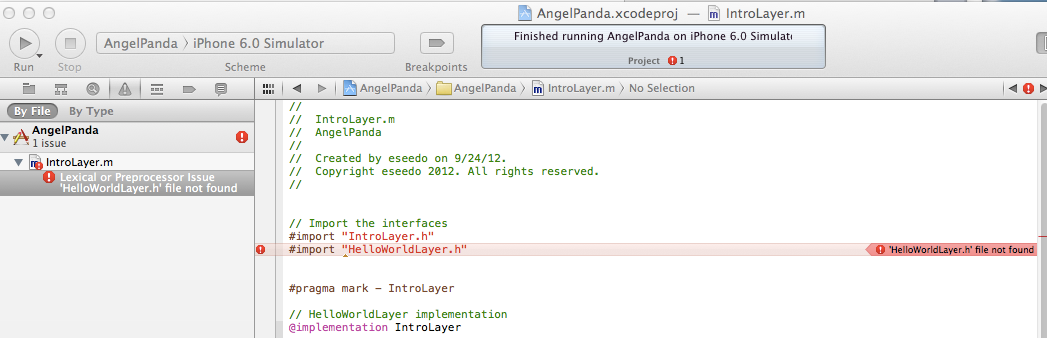
其中scene类方法创建并初始化了一个场景以及ActionLayer层，并将层添加为场景的子节点，并返回场景。

而init方法中，使用CCLabelTTF创建并初始化了一个标签，标签的文本内容是“ANGEL PANDA”，使用的字体是Noteworthy-Bold，文本的字体大小是40。

**注意** 更多iOS字体可以查看 http://iosfonts.com/。

接下来用CCDirector的单例获取了屏幕的大小，并将标签放置在屏幕的中心。最后，将标签添加为当前层的子节点。

此时编译运行项目，会提示以下错误：“HelloWorldLayer.h”file not found，如图10-9所示。



1. 提示错误“HelloWorldLayer.h”file not found

这是因为刚才已经删除了该文件，而在IntroLayer.m中仍然引用了该文件。

此时需要切换到IntroLayer.m，在文件顶部删除以下代码：

#import “HelloWorldLayer.h”

然后添加以下代码：

#import "ActionLayer.h"

在IntroLayer.m中找到onEnter方法，并修改其中的代码如下所示：

-(void) onEnter

{

[super onEnter];

[[CCDirector sharedDirector] replaceScene:[CCTransitionFade transitionWithDuration:1.0 scene:[ActionLayer scene] ]];

}

此时编译运行项目，会看到如图10-10所示场景。



1. 游戏开始画面的当前场景
2. 设置游戏主场景

游戏的主场景如同电影的舞台，所有角色和交互都会在场景中进行。当前的游戏开始界面有点单调，因此首先将给游戏的主场景添加背景。考虑到游戏的需要，这里要添加的不是一个静态的背景图片，而是随着游戏的进展不断滚动的地图。因此这里将使用CCParallaxNode产生视差滚动地图。考虑到之前没有接触过CCParallaxNode，这里需要稍微解释一下它的用法。

CCParallaxNode继承自CCNode，它可以通过addChild:parallaxRatio:positionOffset这个实例方法来添加多个子节点。通过指定CCParallaxNode子节点的运动速度，可以产生类似视差滚动的效果。而视差滚动是指让游戏的多层背景，或者让背景的不同部分以不同的速度移动，形成运动视差的仿3D效果。

在Cocos2D中实现视差滚动并不难，这样按以下三步操作就可以了：

1. 创建一个CCParallaxNode对象，并将其添加到层中。
2. 创建希望产生视差滚动效果的节点对象，并将它们添加为CCParallaxNode的子节点，通常是使用addChild:parallaxRatio:positionOffset方法来实现。
3. 移动CCParallaxNode以滚动背景。根据所设置的parallaxRatio数值，可以让CCParallaxNode的子节点滚动的比背景更快或更慢。

理论知识具备了以后，接下来通过代码实现。

首先切换到ActionLayer.h，增加几个实例变量，更改后的代码如代码清单10-3所示。

1. 添加实例变量

@interface ActionLayer: CCLayer{

CCParallaxNode \*backgroundNode;

CCSprite \*mainBg;

CCSprite \*mainBg2;

CCSprite \*cloud1;

}

以上代码中声明了一个backgroundNode视差滚动节点对象，然后声明了几个精灵对象，后续将会把这几个精灵对象添加为backgroundNode的子节点。

切换到ActionLayer.m，在@implementation ActionLayer之前添加以下代码：

@interface ActionLayer()

//场景初始化

- (void)setupBackground;

@end

注意 从iOS 5.0开始，开发者无需在.h文件中预先声明所有的实例方法，可以在.m文件中的前面进行声明。

在+(id)scene类方法的后面添加setupBackground方法的实现，如代码清单10-4所示。

1. setupBackground方法的实现代码

//添加滚动背景

- (void)setupBackground {

CGSize winSize = [CCDirector sharedDirector].winSize;

// 1) 创建 CCParallaxNode视差滚动节点

backgroundNode = [CCParallaxNode node];

[self addChild:backgroundNode z:-2];

// 2) 创建需要添加到CCParallaxNode视差滚动节点的精灵对象

mainBg = [CCSprite spriteWithFile:@"bg.png"];

mainBg2 =[CCSprite spriteWithFile:@"grass.png"];

cloud1 = [CCSprite spriteWithFile:@"bg\_cloud\_1.png"];

// 3) 设置云彩的浮动速度 和背景速度

CGPoint cloudSpeed = ccp(0.1,0.1);

CGPoint bgSpeed = ccp(0.05, 0.05);

// 4) 将精灵对象添加为CCParallaxNode视差滚动节点的子节点

[backgroundNode addChild:cloud1 z:0 parallaxRatio:cloudSpeed positionOffset:ccp(0,winSize.height\*0.6)];

[backgroundNode addChild:cloud2 z:0 parallaxRatio:cloudSpeed positionOffset:ccp(winSize.width\*0.5,winSize.height\*0.7)];

[backgroundNode addChild:cloud3 z:0 parallaxRatio:cloudSpeed positionOffset:ccp(winSize.width\*0.9,winSize.height\*0.8)];

[backgroundNode addChild:mainBg z:-1

parallaxRatio:bgSpeed

positionOffset:ccp(200,winSize.height\*0.5)];

[backgroundNode addChild:mainBg2 z:0 parallaxRatio:bgSpeed positionOffset:ccp(winSize.width\*0.5,0)];

}

在以上代码中，首先获取屏幕大小，然后使用CCParallaxNode创建视差滚动节点对象。接下来创建需要添加到CCParallaxNode视差滚动节点的精灵对象。

本示例设置两种不同的移动速度，分别代表云彩的精灵对象的移动速度，以及背景的精灵对象的移动速度。最后，将刚才所创建的精灵对象都添加为CCParallaxNode节点的子节点。

在init方法中更改代码如下：

//场景初始化

- (id)init {

if ((self = [super init])) {

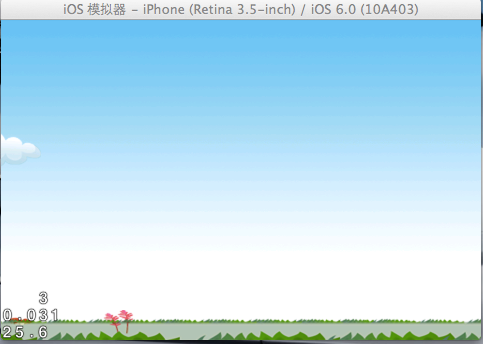
[self setupBackground];

}

return self;

}

此时编译运行项目，可以看到屏幕中出现了背景图片，如图10-11所示。



1. 游戏主场景的背景效果

遗憾的是，此时游戏背景中的云彩并没有浮动。为了让CCParallaxNode节点移动，根据设定的速度每帧更新一次位置，为此需要在ActionLayer.m中对代码做以下修改：

1. 首先在init方法的最后添加以下代码：

[self scheduleUpdate];

1. 在setupBackground方法之后添加新的updateBackground方法，其代码如下：

// 添加新的updateBackground方法

- (void)updateBackground:(ccTime)dt {

CGPoint backgroundScrollVel = ccp(-1000, 0);

backgroundNode.position = ccpAdd(backgroundNode.position, ccpMult(backgroundScrollVel, dt));

}

1. 在init方法之后添加新的update方法，其代码如下：

//实时更新

- (void)update:(ccTime)dt {

[self updateBackground:dt];

}

此时编译运行项目，背景中的云彩开始浮动起来了，但同时又产生了新的问题。不久，背景变得漆黑一片，如图10-12所示。



1. 游戏主场景的视差滚动效果
2. 为了使背景看起来在不停地滚动，方法是：当背景从左侧移出屏幕时将其移动到右侧。在ActionLayer.m中找到updateBackground方法，并更改其代码如下：

//更新背景内容

- (void)updateBackground:(ccTime)dt {

CGSize size = [CCDirector sharedDirector].winSize;

CGPoint backgroundScrollVel = ccp(-size.width, 0);

backgroundNode.position =

ccpAdd(backgroundNode.position,

ccpMult(backgroundScrollVel, dt));

CGSize winSize = [CCDirector sharedDirector].winSize;

NSArray \*backgrounds = [NSArray arrayWithObjects:mainBg, mainBg2, cloud1, cloud2,cloud3, nil];

for (CCSprite \*background in backgrounds) {

if ([backgroundNode convertToWorldSpace:background.position].x < -background.contentSize.width) {

backgroundNode.position = ccp(winSize.width\*4,0);

}

}

}

以上代码中新增的部分代码首先定义了一个数组，然后使用for循环遍历该数组，当视差滚动节点相对于精灵对象的横轴坐标小于精灵对象自身宽度的负值时，就会将视差滚动节点的横向坐标设置为屏幕宽度的4倍位置。之所以设置为屏幕宽度4倍的位置，是因为所选用的背景图片是屏幕宽度的2倍。

此时再次编译项目，游戏主场景的背景就会连续不断地出现在屏幕中了。如图10-13所示。



1. 正常显示的游戏主场景视差滚动效果

此时，游戏主场景的设置工作已完成，接下来将添加游戏中的角色。

1. 添加游戏角色

游戏中的角色包含两部分，分别是代表玩家的角色和代表敌人的角色。

在添加游戏中的玩家角色之前，首先需要添加游戏开始画面的菜单选项。当玩家触碰Play按钮时，才会让玩家角色出现在屏幕之中。

#### 添加菜单选项及玩家角色

首先在Xcode中切换到ActionLayer.h，并在变量声明部分添加以下代码：

CCSprite \*player;

CCLabelBMFont \* titleLabel1;

CCLabelBMFont \* titleLabel2;

CCMenuItemLabel \*playItem;

CCSpriteBatchNode \*batchNode;

以上代码添加了几个精灵对象，分别是代表玩家角色的精灵对象，以及代表游戏开始画面的几个菜单选项，同时还定义了一个精灵表单。

切换到ActionLayer.m，在@interface ActionLayer()的方法声明部分添加以下5个方法的声明：

- (void)playTapped:(id)sender;

-(void)setupTitle;

- (void)invisNode:(CCNode \*)sender;

- (void)setupBatchNode;

-(void)spawnPlayer;

* playTapped：当用户触碰Play按钮时让菜单选项从屏幕中消失，同时调用spawnPlayer方法。
* setupTitle：在屏幕中添加菜单选项。
* invisNode：让节点对象隐藏不再显示。
* setupBatchNode：加载精灵表单。
* spawnPlayer：让代表玩家的角色出现在游戏的主场景中。

在init方法前添加这5个方法的实现代码如代码清单10-5所示。

1. 5个方法的实现

//点按菜单后的操作

- (void)playTapped:(id)sender {

NSArray \* nodes = [NSArray arrayWithObjects:titleLabel1, titleLabel2, playItem, nil];

for (CCNode \*node in nodes) {

[node runAction:

[CCSequence actions:

[CCEaseOut actionWithAction:

[CCScaleTo actionWithDuration:0.5 scale:0] rate:4.0],

[CCCallFuncN actionWithTarget:self selector:@selector(invisNode:)],

nil]];

}

[self spawnPlayer];

}

//让节点对象隐形

- (void)invisNode:(CCNode \*)sender {

sender.visible = FALSE;

}

//设置游戏菜单

- (void)setupTitle {

CGSize winSize = [CCDirector sharedDirector].winSize;

NSString \*fontName = @"Arial.fnt";

titleLabel1 = [CCLabelBMFont labelWithString:@"Angel Panda" fntFile:fontName];

titleLabel1.scale = 0;

titleLabel1.position = ccp(winSize.width/2, winSize.height \* 0.8);

[self addChild:titleLabel1 z:100];

[titleLabel1 runAction:

[CCEaseOut actionWithAction:

[CCScaleTo actionWithDuration:1.0 scale:1.5] rate:4.0]];

titleLabel2 = [CCLabelBMFont labelWithString:@"The Way Home" fntFile:fontName];

titleLabel2.position = ccp(winSize.width/2, winSize.height \* 0.65);

titleLabel2.scale = 0;

[self addChild:titleLabel2 z:100];

[titleLabel2 runAction:

[CCSequence actions:

[CCDelayTime actionWithDuration:1.0],

[CCEaseOut actionWithAction:

[CCScaleTo actionWithDuration:1.0 scale:0.8] rate:4.0],

nil]];

CCLabelBMFont \*playLabel = [CCLabelBMFont labelWithString:@"Play" fntFile:fontName];

playItem = [CCMenuItemLabel itemWithLabel:playLabel target:self selector:@selector(playTapped:)];

playItem.scale = 0;

playItem.position = ccp(winSize.width/2, winSize.height \* 0.5);

CCMenu \*menu = [CCMenu menuWithItems:playItem, nil];

menu.position = CGPointZero;

[self addChild:menu];

[playItem runAction:

[CCSequence actions:

[CCDelayTime actionWithDuration:2.0],

[CCEaseOut actionWithAction:

[CCScaleTo actionWithDuration:0.5 scale:0.8] rate:4.0],

nil]];

}

//添加精灵表单

- (void)setupBatchNode {

batchNode = [CCSpriteBatchNode batchNodeWithFile:@"panda.png"];

[self addChild:batchNode z:-1];

[[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:@"panda.plist"];

}

//让玩家角色出现

- (void)spawnPlayer {

CGSize winSize = [CCDirector sharedDirector].winSize;

player = [CCSprite spriteWithSpriteFrameName:@"panda\_1.png"];

player.position = ccp(-player.contentSize.width/2,

16+player.contentSize.height/2);

[batchNode addChild:player z:1];

[player runAction:

[CCSequence actions:

[CCEaseOut actionWithAction:

[CCMoveBy actionWithDuration:0.5

position:ccp(player.contentSize.width/2 + winSize.width\*0.3, 0)]

rate:4.0],

[CCEaseInOut actionWithAction:

[CCMoveBy actionWithDuration:0.5

position:ccp(-winSize.width\*0.2, 0)]

rate:4.0],

nil]];

NSMutableArray \*animFrames = [NSMutableArray arrayWithCapacity:2];

for(int i=1; i<2;i++){

NSString \*frameName = [NSString stringWithFormat:@"panda\_%d.png",i];

CCSpriteFrame \*frame = [[CCSpriteFrameCache sharedSpriteFrameCache]spriteFrameByName:frameName];

[animFrames addObject:frame];

}

CCAnimation \*animation = [CCAnimation animationWithSpriteFrames:animFrames delay:0.2];

[player runAction:[CCRepeatForever actionWithAction:

[CCAnimate actionWithAnimation:animation]]];

}

这几个方法的代码看起来比较多，但其实都比较简单，接下来依次进行详细解释。

##### playTapped:

该方法首先定义了一个节点数组，并将几个菜单选项添加到数组中。接下来使用for循环遍历节点数组中的所有节点，并让所有的节点都执行一个序列动作（在第2章和第3章中都曾对节点动作进行过详细的说明）。执行这里的序列动作，将让节点对象在0.5秒缩小到原来的一半并淡入，然后让自身调用invisNode:方法，该方法的作用是让节点对象隐藏不再显示。最后会调用spawnPlayer方法让玩家角色出现在屏幕中。

##### invisNode:

该方法让节点对象隐藏不再显示。

##### setupTitle

该方法用于显示游戏主界面的菜单选项。

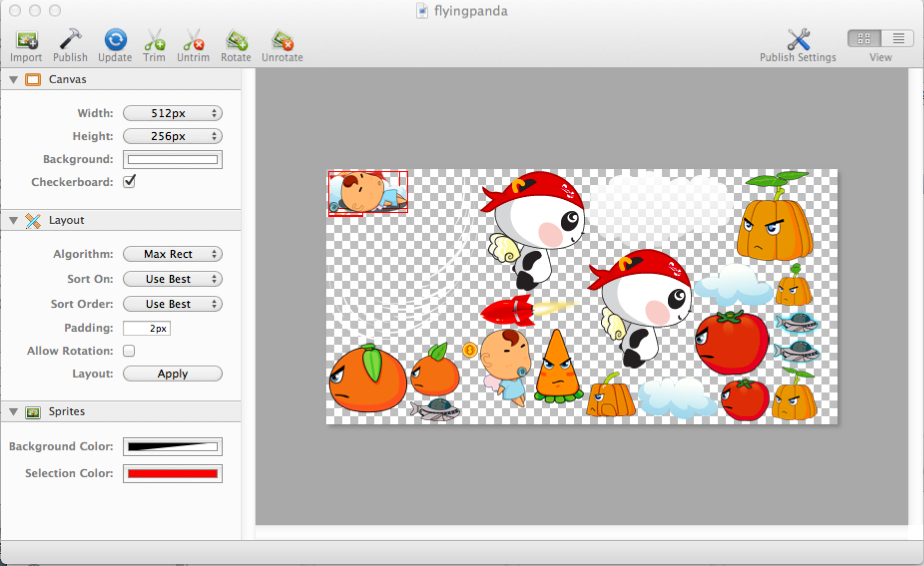
首先使用CCDirector的单例获取屏幕的大小，然后设置一个字体名称。接下来的两段代码很类似，分别添加Angel Panda和The Way Home两个文本标签。

在添加文本标签Angel Panda的代码中，首先使用CCLabelBMFont的类方法labelWithString创建并初始化了一个文本标签。接着设置其比例和在屏幕中的位置，然后将其添加为当前层的子节点。最后让该文本标签执行一个动作，即在1秒内放大到1.5倍大小并淡入。添加文本标签The Way Home的代码与之类似。

接下来添加让用户触碰后调用playTapped方法的标签，为此使用CCMenuItemLabel的类方法itemWithLabel创建并初始化了一个按钮标签。在添加完三个标签后，使用CCMenu的类方法menuWithItems创建了一个菜单，并将Play按钮标签添加为菜单项。最后，让Play按钮标签执行一个序列动作，首先延迟2秒，然后在0.5秒内缩小到原始大小的0.8倍并淡入。

##### setupBatchNode

该方法用于加载精灵表单。AngelPanda游戏中所有游戏角色的图像都保存在使用Zwoptex创建的精灵表单中，如图10-14所示。本书4.7.2节曾详细讲解过如何使用Zwoptex工具制作精灵表单，这里也不再赘述。



1. 保存了所有游戏角色的精灵表单

在setupBatchNode方法中，首先使用CCSpriteBatchNode的类方法batchNodeWithFile创建并初始化一个精灵表单，然后将其添加为当前层的子节点，接着使用CCSpriteFrameCache单例的addSpriteFramesWithFile添加panda.plist中的精灵帧信息。其中panda.png和panda.plist文件是使用Zwoptex工具创建并导出的。

##### spawnPlayer

该方法用于添加玩家角色。

首先使用CCDirector获取屏幕的大小，然后使用CCSprite的spriteWithSpriteFrameName方法从精灵帧中获取精灵图片以创建代表玩家的精灵对象。在设置了玩家角色的初始位置后，将其添加为精灵表单的子节点。

接着，为了让玩家角色的出现更具动态效果，再次使用一个相对复杂的序列动作。在序列动作中，玩家角色将在0.5秒内移动一段距离并淡入，然后在0.5秒内移动一段距离，淡出并淡入。接下来，为了让代表玩家角色的熊猫天使更加可爱，为其添加了一个很简单的动画。

首先创建一个CCSpriteFrameCache精灵帧缓存对象，然后创建并初始化一个CCAnimation动画，为该动画添加两帧，分别是张开翅膀的熊猫天使和收起翅膀的熊猫天使，并设置延迟0.2秒。最后，使用CCRepeatForever让熊猫天使持续运行该动画，从而让游戏角色更生动逼真。

在init方法中添加以下代码：

[self setupBatchNode];

[self setupTitle];

编译运行项目，首先会看到游戏的主场景中出现菜单选项，如图10-15所示。



1. 游戏菜单选项

触碰Play按钮时，文本标签和菜单项会从屏幕中消失，而熊猫天使会从屏幕左侧出现，如图10-16所示。



1. 熊猫天使出现在屏幕中

至此，游戏中的玩家角色熊猫天使已经创建完毕。

#### 添加敌人角色

虽然说添加游戏中的各种敌人，但实际上将要添加两类角色，一类是给玩家带来伤害的真正敌人，而另一类则是给玩家带来奖励分值和疗伤的游戏帮手。因为它们在游戏中的添加方式相似，因此统称为游戏中的敌人角色。

由于这些角色在游戏中将不断产生和消失，而每一次创建新角色时都需要占用一定的内存。为了方便起见，这里将使用一个辅助方法管理敌人角色数组。

在Xcode中右键单击Angry Panda组，选择New File，然后选择iOS→Cocoa Touch→Objective-C class，选择Next。选择Subclass of NSObject，单击Next，将文件命名为SpriteArray.m，单击Save。

1. 切换到SpriteArray.h，使用以下代码替代其中的内容：

#import "cocos2d.h"

@interface SpriteArray : NSObject {

CCArray \* \_array;

int \_nextItem;

}

@property (readonly) CCArray \* array;

- (id)initWithCapacity:(int)capacity spriteFrameName:(NSString \*)spriteFrameName batchNode:(CCSpriteBatchNode \*)batchNode;

- (id)nextSprite;

@end

在以上代码中，首先声明了SpriteArray类，它直接继承自NSObject。同时声明了两个实例变量，一个用于保存精灵对象的数组，而另一个代表下一个精灵对象的数字编号。此外，用于保存精灵对象的数组被声明为属性，以便在游戏的主场景中调用。最后，声明了两个实例方法，分别用于创建精灵数组和获取下一个精灵对象。

1. 在Xcode中切换到SpriteArray.m，并使用以下代码替代其中的内容：

#import "SpriteArray.h"

@implementation SpriteArray

@synthesize array = \_array;

- (id)initWithCapacity:(int)capacity spriteFrameName:(NSString \*)spriteFrameName batchNode:(CCSpriteBatchNode \*)batchNode {

if ((self = [super init])) {

\_array =

[[CCArray alloc] initWithCapacity:capacity];

for(int i = 0; i < capacity; ++i) {

CCSprite \*sprite = [CCSprite

spriteWithSpriteFrameName:spriteFrameName];

sprite.visible = NO;

[batchNode addChild:sprite];

[\_array addObject:sprite];

}

}

return self;

}

- (id)nextSprite {

id retval = [\_array objectAtIndex:\_nextItem];

\_nextItem++;

if (\_nextItem >= \_array.count) \_nextItem = 0;

return retval;

}

- (void)dealloc {

[\_array release];

\_array = nil;

[super dealloc];

}

@end

在以上代码中，主要是实现了刚才所声明的两个实例方法。

其中在initWithCapacity方法的实现代码中，使用CCArray创建并初始化了一个节点数组。然后使用for循环为该数组添加精灵对象。最后返回所生成的数组。nextSprite方法相对简单，它使用CCArray的objectAtIndex方法获取下一个精灵对象，并返回所获取的对象。最后，dealloc方法主要是释放内存。

1. 切换到ActionLayer.h，首先在文件的顶部添加以下代码：

#import "SpriteArray.h"

在类声明中添加以下实例变量的声明：

SpriteArray \*missleArray;

SpriteArray \*groundObjectsArray;

SpriteArray \*airObjectsArray;

SpriteArray \*coinsArray;

SpriteArray \*aidArray;

以上代码添加几个SpriteArray类型的变量，分别代表攻击玩家的导弹、地面怪物、从空中攻击玩家的飞机、对玩家的金币奖励和对玩家提供疗伤的医疗包。

同时还要添加几个变量声明：

double nextMissleSpawn;

double nextGroundObstacleSpawn;

double nextAirObstacleSpawn;

double nextCoinSpawn;

double nextAid;

double curTime;

以上代码所声明的变量前五个用于从数组中获取下一个精灵对象，而curTime则用于获取游戏中的当前时间。

1. 切换到ActionLayer.m，并在方法声明部分添加以下代码：

- (void)setupArrays;  
-(void)updateGroundObstacles:(ccTime)dt;

-(void)updateAirObstacles:(ccTime)dt;

-(void)updateCoins:(ccTime)dt;

- (void)updateMissles:(ccTime)dt;

-(void)updateAid:(ccTime)dt;

以上方法中，setupArrays用于初始化精灵数组，而其余几个方法分别用于在场景中添加地面怪物，空中攻击玩家的飞机，金币，导弹和急救包。

1. 在init方法前添加以上方法的实现代码如代码清单10-6所示。
2. 实现代码

//添加随机函数

float randomValueBetween(float low, float high) {

return (((float) arc4random() / 0xFFFFFFFFu)

\* (high - low)) + low;

}

//初始化精灵数组

- (void)setupArrays {

missleArray = [[SpriteArray alloc] initWithCapacity:30 spriteFrameName:@"missle.png" batchNode:batchNode];

groundObjectsArray = [[SpriteArray alloc] initWithCapacity:30 spriteFrameName:@"carrot.png" batchNode:batchNode];

airObjectsArray = [[SpriteArray alloc] initWithCapacity:30 spriteFrameName:@"plane3.png" batchNode:batchNode];

coinsArray = [[SpriteArray alloc] initWithCapacity:30 spriteFrameName:@"coin.png" batchNode:batchNode];

aidArray =[[SpriteArray alloc] initWithCapacity:30 spriteFrameName:@"aid.png" batchNode:batchNode];

}

//添加地面怪物

-(void)updateGroundObstacles:(ccTime)dt{

CGSize winSize = [CCDirector sharedDirector].winSize;

// 判断是否是添加地面怪物的时间？

curTime = CACurrentMediaTime();

if (curTime > nextGroundObstacleSpawn) {

// 确认下一次出现地面怪物的时间

float randSecs = randomValueBetween(3.0, 5.0);

nextGroundObstacleSpawn = randSecs + curTime;

// 计算一个让怪物穿越屏幕的随机时间

float randDuration = randomValueBetween(1.5, 2.0);

// 创建一个新的怪物精灵

CCSprite \*groundObject = [groundObjectsArray nextSprite];

[groundObject stopAllActions];

groundObject.visible = YES;

// 设置其初始位置

groundObject.position = ccp(winSize.width+groundObject.contentSize.width/2, groundObject.contentSize.height/2);

// 设置一个随机大小

int randNum = arc4random() % 3;

if (randNum == 0) {

groundObject.scale = 0.5;

} else if (randNum == 1) {

groundObject.scale = 0.8;

} else {

groundObject.scale = 1.2;

}

id delayTime = [CCDelayTime actionWithDuration:4.0f];

[groundObject runAction:

[CCSequence actions:

delayTime,

[CCMoveBy actionWithDuration:randDuration position:ccp(-winSize.width-groundObject.contentSize.width, 0)],

[CCCallFuncN actionWithTarget:self selector:@selector(invisNode:)],

nil]];

}

}

//添加空中攻击玩家的飞机

-(void)updateAirObstacles:(ccTime)dt{

CGSize winSize = [CCDirector sharedDirector].winSize;

// 判断是否是出现飞机的时间

curTime = CACurrentMediaTime();

if (curTime > nextAirObstacleSpawn) {

// 确认下一次出现飞机的时间

float randSecs = randomValueBetween(3.0, 5.0);

nextAirObstacleSpawn = randSecs + curTime;

// 计算一个随机的Y坐标

float randY = randomValueBetween(0.0,

winSize.height);

// 计算一个飞机横穿屏幕的随机时间

float randDuration = randomValueBetween(1.5, 2.0);

// 创建一个新的飞机精灵

CCSprite \*airObject = [airObjectsArray nextSprite];

[airObject stopAllActions];

airObject.visible = YES;

// 设置初始位置

airObject.position = ccp(winSize.width+airObject.contentSize.width/2, randY);

// 设置随机大小

int randNum = arc4random() % 3;

if (randNum == 0) {

airObject.scale = 0.5;

} else if (randNum == 1) {

airObject.scale = 0.8;

} else {

airObject.scale = 1.0;

}

// 让其穿过屏幕，然后消失

id delayTime = [CCDelayTime actionWithDuration:4.0f];

[airObject runAction:

[CCSequence actions:

delayTime,

[CCMoveBy actionWithDuration:randDuration position:ccp(-winSize.width-airObject.contentSize.width, 0)],

[CCCallFuncN actionWithTarget:self selector:@selector(invisNode:)],

nil]];

}

}

//添加奖赏玩家的金币

-(void)updateCoins:(ccTime)dt{

CGSize winSize = [CCDirector sharedDirector].winSize;

// 判断是否该出现金币？

curTime = CACurrentMediaTime();

if (curTime > nextCoinSpawn) {

// 计算下一次出现金币的随机时间

float randSecs = randomValueBetween(3.0, 6.0);

nextCoinSpawn = randSecs + curTime;

// 计算一个随机的y坐标

float randY = randomValueBetween(winSize.height/2,

winSize.height-100);

float randCoinNumber = randomValueBetween(2.0, 5.0);

// int randNumber =int(randCoinNumber);

// 计算金币穿越屏幕的随机时间

float randDuration = randomValueBetween(3.5, 5.0);

// 创建一堆金币

for (int i=1;i<=randCoinNumber;i++)

for (int j=1;j<=randCoinNumber;j++){

CCSprite \*coins = [coinsArray nextSprite];

[coins stopAllActions];

coins.visible = YES;

// 设置其初始位置

coins.position = ccp(winSize.width+coins.contentSize.width/2+30\*i, randY+30\*j);

// 让金币穿越屏幕，然后消失

[coins runAction:

[CCSequence actions:

[CCMoveBy actionWithDuration:randDuration position:ccp(-winSize.width-coins.contentSize.width, 0)],

[CCCallFuncN actionWithTarget:self selector:@selector(invisNode:)],

nil]];

}

}

}

//添加攻击玩家的导弹

- (void)updateMissles:(ccTime)dt {

CGSize winSize = [CCDirector sharedDirector].winSize;

BOOL firstSpawn;

// 判断是否是出现导弹的时间

curTime = CACurrentMediaTime();

if (curTime > nextMissleSpawn) {

// 计算下一次出现导弹的时间

float randSecs = randomValueBetween(15.0, 18.0);

nextMissleSpawn = randSecs + curTime;

// 计算一个随机的y坐标

float randY = randomValueBetween(0.0,

winSize.height);

// 计算让导弹飞过屏幕的随机时间

float randDuration = randomValueBetween(0.3, 0.5);

// 创建一个新的导弹精灵

CCSprite \*missle = [missleArray nextSprite];

[missle stopAllActions];

missle.visible = YES;

// 设置其初始位置

missle.position = ccp(winSize.width+missle.contentSize.width/2, randY);

// 设置其随机大小

int randNum = arc4random() % 3;

if (randNum == 0) {

missle.scale = 0.5;

} else if (randNum == 1) {

missle.scale = 0.7;

} else {

missle.scale = 0.8;

}

if(firstSpawn!=NO){

//play sound

}

// 让导弹穿过屏幕然后消失

id delayTime = [CCDelayTime actionWithDuration:4.0f];

[missle runAction:

[CCSequence actions:

delayTime,

[CCMoveBy actionWithDuration:randDuration position:ccp(-winSize.width-missle.contentSize.width, 0)],

[CCCallFuncN actionWithTarget:self selector:@selector(invisNode:)],

nil]];

}

firstSpawn = YES;

}

//添加对玩家提供疗伤的急救包

- (void)updateAid:(ccTime)dt {

CGSize winSize = [CCDirector sharedDirector].winSize;

// 判断是否出现急救包

curTime = CACurrentMediaTime();

if (curTime > nextAid) {

// 计算下一次出现急救包的时间

float randSecs = randomValueBetween(18.0, 20.0);

nextAid = randSecs + curTime;

// 计算一个随机的y坐标

float randY = randomValueBetween(0.0,

winSize.height);

// 计算急救包穿过屏幕的随机时间

float randDuration = randomValueBetween(2.5, 4.0);

// 创建一个新的急救包精灵

CCSprite \*aid = [aidArray nextSprite];

[aid stopAllActions];

aid.visible = YES;

// 设置其初始位置

aid.position = ccp(winSize.width+aid.contentSize.width/2, randY);

// 设置其随机大小

int randNum = arc4random() % 3;

if (randNum == 0) {

aid.scale = 0.45;

} else if (randNum == 1) {

aid.scale = 0.65;

} else {

aid.scale = 0.9;

}

// 让急救包穿过屏幕然后消失

id delayTime = [CCDelayTime actionWithDuration:4.0f];

[aid runAction:

[CCSequence actions:

delayTime,

[CCMoveBy actionWithDuration:randDuration position:ccp(-winSize.width-aid.contentSize.width, 0)],

[CCCallFuncN actionWithTarget:self selector:@selector(invisNode:)],

nil]];

}

}

在各方法的实现代码前，添加了一个randomValueBetween随机函数，用于获取两个数值间的随机值。

接下来看setupArrays方法的实现代码，其中调用SpriteArray的实例方法创建并初始化了几个精灵数组，分别代表导弹，地面怪物，空中的飞机，金币和急救包。

接下来的几个方法非常类似，这里只详细解释用于添加导弹的updateMissles方法。

在该方法的实现代码中，首先使用CCDirector的单例获取屏幕大小。然后定义一个布尔变量用于判断导弹是否第一次出现在屏幕中。使用CACurrentMediaTime()函数获取游戏中的当前时间。然后使用if判断，如果当前时间大于下一个导弹的出现时间，则使用随机函数生成一个randSecs随机时间间隔，并设置下一个导弹的出现时间为当前时间加上随机时间间隔。接下来再次使用随机函数生成一个随机数，作为导弹从屏幕中飞行的时间。然后从刚才创建的导弹数组中获取一个导弹精灵，停止其所有的动作，设置其可见，同时设置其在屏幕中的位置。接下来使用if判断来确定导弹的显示比例。此时，如果并非第一次出现，则会随后播放音效，其代码将在后续添加。最后，让导弹执行一个序列动作，即首先等待4秒，然后在刚才设置的随机时间内从屏幕右侧飞行到左侧，最后让自身从屏幕中消失。

其他几个方法的实现代码和该方法几乎完全相同，这里不再赘述。

1. 在添加完以上方法的实现代码后，还需要在update方法中添加以下代码：

[self updateMissles:dt];

[self updateAirObstacles:dt];

[self updateGroundObstacles:dt];

[self updateCoins:dt];

[self updateAid:dt];

1. 还需要在init方法中添加以下代码：

[self setupArrays];

此时编译运行项目，各种或敌或友的角色就会出现在游戏的主场景中。如图10-17所示。



1. 代表敌人的角色出现在屏幕中

至此，已经成功的在游戏场景中添加了代表敌人的角色。下一节将添加玩家和游戏交互的处理机制。

1. 添加用户交互机制

现在的游戏场景已经非常生动，但任何一款游戏都需要玩家的互动才会具有游戏性。在AngelPanda游戏中，主要的互动方式将模仿Jetpack Joyride，即玩家需要触碰屏幕在场景中穿行，并避开各种危险的攻击，同时尽量获得金币奖励，以及急救包的治疗。

在Xcode中切换到ActionLayer.h，在类声明部分添加以下代码：

float playerPointsPerSecY;

以上代码只是声明了一个代表熊猫天使在y轴上移动速度的变量。

在Xcode中切换到ActionLayer.m，首先在方法声明部分添加以下代码：

- (void)updatePlayerPos:(ccTime)dt;

在init方法下面添加以下几个方法：

//使用触摸事件来控制熊猫天使

- (void)ccTouchesBegan:(NSSet \*)touches withEvent:(UIEvent \*)event {

playerPointsPerSecY =100.0f;

}

-(void)ccTouchesEnded:(NSSet \*)touches withEvent:(UIEvent \*)event{

playerPointsPerSecY =-150.0f;

}

//更新熊猫天使的位置

- (void)updatePlayerPos:(ccTime)dt {

CGSize winSize = [CCDirector sharedDirector].winSize;

float maxY = winSize.height - player.contentSize.height/2;

float minY = 15+player.contentSize.height/2;

float newY = player.position.y + (playerPointsPerSecY \* dt);

newY = MIN(MAX(newY, minY), maxY);

player.position = ccp(player.position.x, newY);

}

在第5章中曾详细讲解过Cocos2D对触碰事件的处理机制。这里使用的是标准协议代理机制，在ccTouchesBegan方法中，当触碰开始时，让熊猫天使在y轴的移动速度设置为100.0f。而在ccTouchesEnded方法中，当触碰结束时，让熊猫天使在y轴的移动速度设置为-150.0f。如果这里直接将熊猫天使的位置设置为在y轴增加或减少一定的数值，并不会得到很好的效果。因此还需要通过updatePlayerPos方法每帧更新该位置。

在updatePlayerPos方法中，首先获取屏幕的大小，然后设置了熊猫天使在y轴的最高点和最低点（因为场景中有草地）。最后将熊猫天使在y轴的新位置设置为当前位置加上位置的变动。

最后，别忘了在update方法中添加一行代码：

[self updatePlayerPos:dt];

以及在init方法中添加一行代码：

self.isTouchEnabled = YES;

此时编译运行项目，就可以用手指触碰（设备）或鼠标点击（模拟器）控制熊猫天使的移动了。如图10-18所示。



1. 在模拟器中用鼠标控制熊猫天使的移动

至此，已经成功地实现游戏中的交互处理机制，接下来将添加游戏的逻辑和赢输机制。

1. 添加游戏逻辑

任何一款游戏都需要有逻辑机制和赢输判断，在这个小游戏中，只要玩家可以操控熊猫天使成功飞行1分钟而仍然存活，就算胜利。

在游戏中，当导弹、飞机和地面怪物碰到熊猫天使后，它的生命值会有不同程度的下降。当熊猫天使碰到金币时，会得到金币奖励，而当它碰到急救包时，生命值则会上升。为此，将需要在游戏中添加简单的碰撞检测和赢输判断。

1. 在Xcode中切换到ActionLayer.h，在类声明部分添加以下代码：

int lives;

int score;

double gameOverTime;

bool gameOver;

在以上代码中所声明的变量分别代表熊猫天使的生命值、玩家的得分、游戏结束的时间和玩家是否赢得了游戏。

1. 在@interface ActionLayer:CCLayer的前面添加以下代码：

typedef enum {

kEndReasonWin,

kEndReasonLose

} EndReason;

这里只是声明了一个枚举变量，用于说明游戏结束的原因。

1. 切换到ActionLayer.m，在方法声明部分添加以下代码：

//碰撞与游戏逻辑

-(void)updateGameStatus:(ccTime)dt;

-(void)setupTime;

- (void)updateCollisions:(ccTime)dt;

- (void)restartTapped:(id)sender;

- (void)endScene:(EndReason)endReason;

以上方法中，updateGameStatus方法用于实时更新游戏状态，setupTime用于设置游戏的初始值，updateCollisions方法用于基本的碰撞检测，restartTapped方法是重新启动游戏，而endScene方法根据不同的游戏结束原因显示结果。

1. 在init方法前实现这些方法的代码如下：

-(void)updateGameStatus:(ccTime)dt{

if (lives <= 0) {

[player stopAllActions];

player.visible = FALSE;

[self endScene:kEndReasonLose];

} else if (curTime >= gameOverTime) {

[self endScene:kEndReasonWin];

}

}

-(void)setupTime{

lives = 10;

curTime = CACurrentMediaTime();

gameOverTime = curTime + 30.0;

}

//简单的碰撞检测

- (void)updateCollisions:(ccTime)dt {

for (CCSprite \*coins in coinsArray.array) {

if (!coins.visible) continue;

if(CGRectIntersectsRect(coins.boundingBox, player.boundingBox)){

coins.visible = NO;

score +=100;

}

}

for (CCSprite \*missle in missleArray.array){

if(!missle.visible)continue;

if(CGRectIntersectsRect(missle.boundingBox, player.boundingBox)){

missle.visible = NO;

[player runAction:[CCBlink actionWithDuration:1.0 blinks:9]];

lives-=2;

}

}

for (CCSprite \*groundObject in groundObjectsArray.array){

if(!groundObject.visible)continue;

if(CGRectIntersectsRect(groundObject.boundingBox, player.boundingBox)){

groundObject.visible = NO;

[player runAction:[CCBlink actionWithDuration:1.0 blinks:9]];

lives--;

}

}

for (CCSprite \*airObject in airObjectsArray.array){

if(!airObject.visible)continue;

if(CGRectIntersectsRect(airObject.boundingBox, player.boundingBox)){

airObject.visible = NO;

[player runAction:[CCBlink actionWithDuration:1.0 blinks:9]];

lives--;

}

}

for (CCSprite \*aid in aidArray.array){

if(!aid.visible)continue;

if(CGRectIntersectsRect(aid.boundingBox, player.boundingBox)){

aid.visible = NO;

// [player runAction:[CCBlink actionWithDuration:1.0 blinks:9]];

lives++;

}

}

}

//重启游戏

- (void)restartTapped:(id)sender {

[[CCDirector sharedDirector] replaceScene:[CCTransitionZoomFlipX transitionWithDuration:0.5 scene:[ActionLayer scene]]];

}

//游戏结束画面

- (void)endScene:(EndReason)endReason {

if (gameOver) return;

gameOver = true;

CGSize winSize = [CCDirector sharedDirector].winSize;

NSString \*message;

if (endReason == kEndReasonWin) {

message = @"You win!";

} else if (endReason == kEndReasonLose) {

message = @"You lose!";

}

CCLabelBMFont \*label;

label = [CCLabelBMFont labelWithString:message fntFile:@"Arial.fnt"];

label.scale = 0.1;

label.position = ccp(winSize.width/2, winSize.height \* 0.6);

[self addChild:label];

CCLabelBMFont \*restartLabel;

restartLabel = [CCLabelBMFont labelWithString:@"Restart" fntFile:@"Arial.fnt"];

CCMenuItemLabel \*restartItem = [CCMenuItemLabel itemWithLabel:restartLabel target:self selector:@selector(restartTapped:)];

restartItem.scale = 0.1;

restartItem.position = ccp(winSize.width/2, winSize.height \* 0.4);

CCMenu \*menu = [CCMenu menuWithItems:restartItem, nil];

menu.position = CGPointZero;

[self addChild:menu];

[restartItem runAction:[CCScaleTo actionWithDuration:0.5 scale:1.0]];

[label runAction:[CCScaleTo actionWithDuration:0.5 scale:1.0]];

}

以下将分别解释各个方法的作用。

##### updateGameStatus

该方法的实现代码中，如果熊猫天使的生命值为0，则让游戏角色停止一切动作，同时调用endScene方法，并传递参数为kEndReasonLose。而如果游戏时间超过了设置的游戏结束时间，则说明玩家赢得了游戏，此时也将调用endScene方法，并传递参数为kEndReasonWin。

##### setupTime

该方法很简单，在其中设置了熊猫天使的初始生命值为10，游戏结束时间为当前时间再加上30秒。

##### updateCollisions

该方法的代码内容很多，但并不难理解。其中调用了5次for 循环。

以第一个循环为例，首先遍历了金币数组中的所有金币，如果金币不可见，则不进行任何操作。此时，如果金币精灵对象的外框和熊猫天使精灵对象的外框相交，则表示二者发生了碰撞，此时金币将消失，玩家将得到100分的奖励。

在第二个循环中对导弹和熊猫天使的碰撞进行了判断，如果发生了碰撞，除了让导弹消失，玩家自身将在1秒内闪动9次，同时生命值会减少2。

其他几个循环中的内容基本相似。

##### restartTapped

作用是让游戏场景在0.5秒内跳转到ActionLayer的初始场景。

##### endScene

该方法首先使用gameOver布尔变量进行判断游戏是否结束。接下来获取屏幕的大小，并定义了一个字符串变量。如果游戏结束的原因是kEndReasonWin，即赢得游戏，则设置该字符串为“You win!”，反之则设置为“You lose!”。

接下来创建一个文本标签在屏幕中显示游戏的结果。然后创建一个Restart菜单项，当玩家触碰该菜单项时会调用restartTapped方法。最后创建一个菜单，并将Restart菜单项添加到菜单中，然后将菜单添加为当前层的子节点。最后让文本标签和Restart菜单项各自执行一个动作。其中Restart菜单项在0.5秒内缩放到原来的大小，而文本标签也同样如此。

当然别忘了在update方法中添加以下代码：

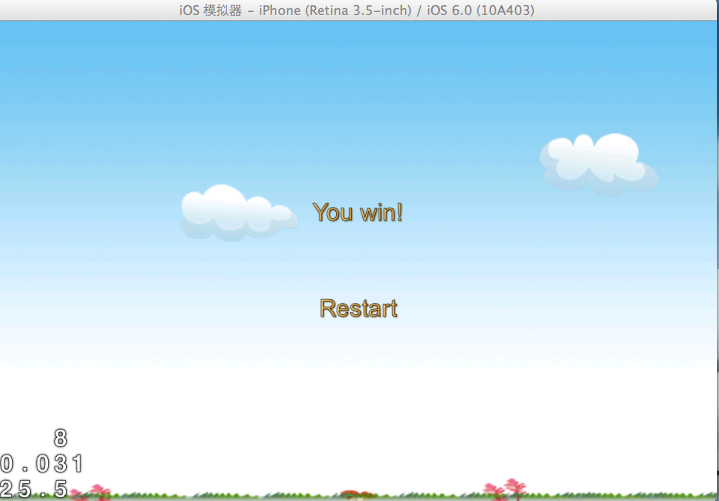
[self updateGameStatus:dt];

[self updateCollisions:dt];

以及在init方法中添加以下代码：

[self setupTime];

编译运行游戏，当游戏成功结束时，将显示“You win!”，如图10-19所示；反之显示“You lose”，如图10-20所示。



1. 赢得游戏的画面



1. 游戏失败的画面

至此，游戏的逻辑机制已经添加成功，只需添加一些音效就大功告成了。

1. 添加音效

没有音效的游戏总让人觉得不太完整，因此，本节将介绍如何添加背景音乐和音效。

1. 在Xcode中切换到ActionLayer.m，在文件的顶部添加以下代码：

#import "SimpleAudioEngine.h"

1. 在方法声明部分添加以下代码：

-(void)setupSound;

1. 在init方法的前面添加以下代码设置声音：

//设置声音

-(void)setupSound{

[[SimpleAudioEngine sharedEngine] playBackgroundMusic:@"begin.mp3" loop:YES];

[[SimpleAudioEngine sharedEngine] preloadEffect:@"angry\_girl.mp3"];

[[SimpleAudioEngine sharedEngine] preloadEffect:@"babylaugh.mp3"];

[[SimpleAudioEngine sharedEngine] preloadEffect:@"babycry.mp3"];

[[SimpleAudioEngine sharedEngine] preloadEffect:@"mother.mp3"];

[[SimpleAudioEngine sharedEngine] preloadEffect:@"coin.mp3"];

}

以上代码中使用SimpleAudioEngine的单例来播放背景音乐，同时预先加载了几个音效文件。关于在Cocos2D游戏中添加音效和音乐的更多知识，可以查看本书第7章。

1. 更改updateCollisions方法的代码如下：

//简单的碰撞检测

- (void)updateCollisions:(ccTime)dt {

for (CCSprite \*coins in coinsArray.array) {

if (!coins.visible) continue;

if(CGRectIntersectsRect(coins.boundingBox, player.boundingBox)){

coins.visible = NO;

score +=100;

[[SimpleAudioEngine sharedEngine]playEffect:@"coin.mp3" pitch:1.0f pan:0.0f gain:0.25f];

}

}

for (CCSprite \*missle in missleArray.array){

if(!missle.visible)continue;

if(CGRectIntersectsRect(missle.boundingBox, player.boundingBox)){

[[SimpleAudioEngine sharedEngine] playEffect:@"angry\_girl.mp3" pitch:1.0f pan:0.0f gain:0.25f];

missle.visible = NO;

[player runAction:[CCBlink actionWithDuration:1.0 blinks:9]];

lives-=2;

}

}

for (CCSprite \*groundObject in groundObjectsArray.array){

if(!groundObject.visible)continue;

if(CGRectIntersectsRect(groundObject.boundingBox, player.boundingBox)){

[[SimpleAudioEngine sharedEngine] playEffect:@"angry\_girl.mp3" pitch:1.0f pan:0.0f gain:0.25f];

groundObject.visible = NO;

[player runAction:[CCBlink actionWithDuration:1.0 blinks:9]];

lives--;

}

}

for (CCSprite \*airObject in airObjectsArray.array){

if(!airObject.visible)continue;

if(CGRectIntersectsRect(airObject.boundingBox, player.boundingBox)){

[[SimpleAudioEngine sharedEngine] playEffect:@"angry\_girl.mp3" pitch:1.0f pan:0.0f gain:0.25f];

airObject.visible = NO;

[player runAction:[CCBlink actionWithDuration:1.0 blinks:9]];

lives--;

}

}

for (CCSprite \*aid in aidArray.array){

if(!aid.visible)continue;

if(CGRectIntersectsRect(aid.boundingBox, player.boundingBox)){

[[SimpleAudioEngine sharedEngine] playEffect:@"mother.mp3" pitch:1.0f pan:0.0f gain:0.25f];

aid.visible = NO;

// [player runAction:[CCBlink actionWithDuration:1.0 blinks:9]];

lives++;

}

}

}

以上代码的变化之处在于，每次发生碰撞时，都会播放对应的音效。

1. 最后更改endScene方法的代码如代码清单10-7所示。
2. 更改endScene方法的代码

//游戏结束画面

- (void)endScene:(EndReason)endReason {

if (gameOver) return;

gameOver = true;

CGSize winSize = [CCDirector sharedDirector].winSize;

NSString \*message;

if (endReason == kEndReasonWin) {

[[SimpleAudioEngine sharedEngine]playEffect:@"babylaugh.mp3"];

message = @"You win!";

} else if (endReason == kEndReasonLose) {

message = @"You lose!";

[[SimpleAudioEngine sharedEngine]playEffect:@"babycry.mp3"];

}

CCLabelBMFont \*label;

label = [CCLabelBMFont labelWithString:message fntFile:@"Arial.fnt"];

label.scale = 0.1;

label.position = ccp(winSize.width/2, winSize.height \* 0.6);

[self addChild:label];

CCLabelBMFont \*restartLabel;

restartLabel = [CCLabelBMFont labelWithString:@"Restart" fntFile:@"Arial.fnt"];

CCMenuItemLabel \*restartItem = [CCMenuItemLabel itemWithLabel:restartLabel target:self selector:@selector(restartTapped:)];

restartItem.scale = 0.1;

restartItem.position = ccp(winSize.width/2, winSize.height \* 0.4);

CCMenu \*menu = [CCMenu menuWithItems:restartItem, nil];

menu.position = CGPointZero;

[self addChild:menu];

[restartItem runAction:[CCScaleTo actionWithDuration:0.5 scale:1.0]];

[label runAction:[CCScaleTo actionWithDuration:0.5 scale:1.0]];

}

在以上代码中，当玩家赢得游戏时，会播放babylaugh这个音效；而如果输了，则会播放babycry这个音效。

最后别忘了在init方法中添加以下代码：

[self setupSound];

编译运行游戏，卡通而又俏皮的音乐会伴随着游戏的进程出现。

至此，一个简单而完整的游戏已经大功告成了！

## 本章小结

本章首先介绍iOS游戏设计与开发的流程；接下来详细介绍了示例游戏Angel Panda的设计，包括创意来源、故事背景、美术和音效风格设计、游戏操作流程、游戏规则设计、关卡设计、开发工具和技术等。为自己独立开发和设计游戏奠定基础。

Angel Panda虽然只是一个简单的示例游戏，但其中几乎涵盖了基础篇（除物理引擎外）的所有内容。第3章的核心类是该游戏的基础；在主场景的开始画面中用到了文本渲染系统来显示菜单和标签；在代表玩家角色出现的部分使用了第4章的动作与动画知识；在游戏的主场景中使用触摸事件处理机制来处理用户交互。最后，使用了第7章的知识给游戏添加背景音乐和音效。通过学习该案例，可以全面掌握之前所学习的Cocos2D知识，并将其付诸实际。

第11章将从零开始一步步带领读者学习如何设计和实现相对复杂，也更为完整的Angry Panda游戏。